Office de la Propriété Intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An agency of Industry Canada

10/718,809
RD-8912

CA 2249080 C 2001/12/04

(11)(21) 2 249 080

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(22) Date de dépôt/Filing Date: 1998/09/25

(41) Mise à la disp. pub./Open to Public Insp.: 2000/03/25

(45) Date de délivrance/Issue Date: 2001/12/04

(51) Cl.Int.^{6}/Int.Cl.^{6} G06F 17/30

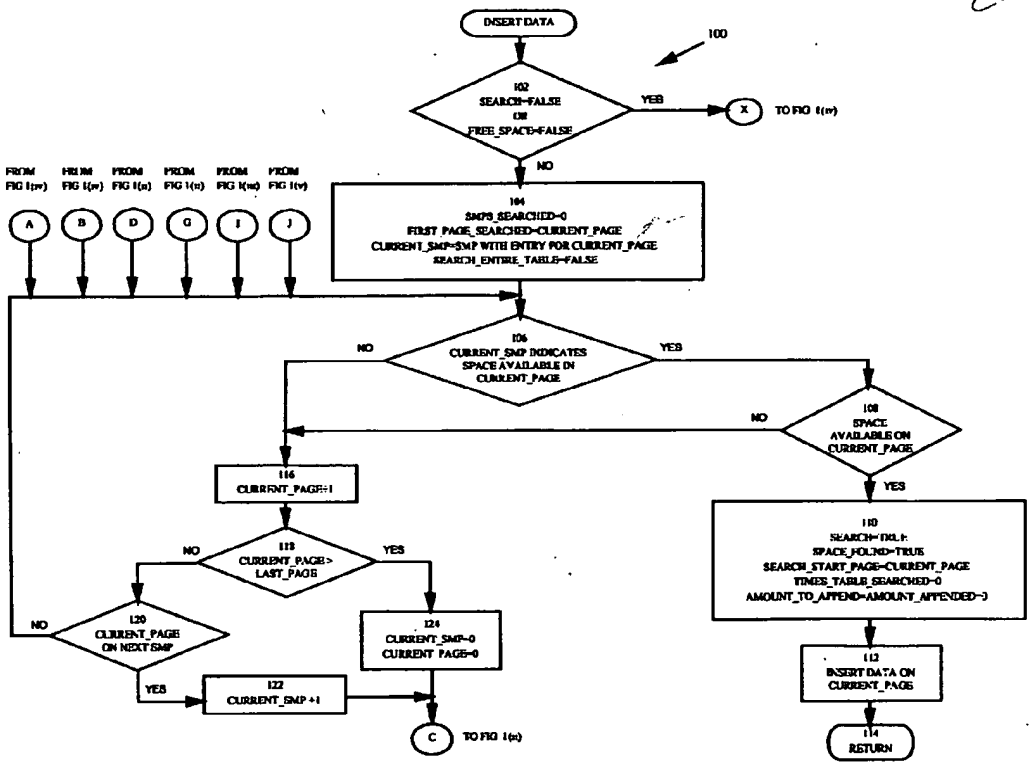
(72) Inventeurs/Inventors:
HOP HING, Nelson, CA;
GOSS, JEFFREY J., CA;
ROMANUFA, KERILEY K., CA;
HURAS, MATTHEW A., CA;
WINER, MICHAEL J., CA;
LINDSAY, Bruce G., US

(73) Propriétaire/Owner:
IBM CANADA LIMITED-IBM CANADA LIMITEE, CA

(74) Agent: ROSEN, ARNOLD

(54) Titre : METHODE DE RECHERCHE EFFICACE D'ESPACE LIBRE DANS UNE BASE DE DONNEES RELATIONNELLES

(54) Title: METHOD FOR EFFICIENTLY SEARCHING FREE SPACE IN A RELATIONAL DATABASE



(57) Abrégé/Abstract:
A method for efficiently searching free space in a relational database management system. The method limits the search to a finite number of space map pages in the free space map. If the configured number of space map pages in the free space map are searched and a page with free space is not found for the row, the row is inserted on the last page, or if no space is available on the last page a new page is created and the row is inserted on the new page. New rows are then appended until some predefined amount of space is filled before a search is done again. As a result, insertion of a row into the database management system does not incur the worst-case cost of searching the entire free space map.

CA9-98-036

ABSTRACT OF THE DISCLOSURE

METHOD FOR EFFICIENTLY SEARCHING FREE SPACE IN A RELATIONAL DATABASE

A method for efficiently searching free space in a relational database management system. The method limits the search to a finite number of space map pages in the free space map. If the configured number of space map pages in the free space map are searched and a page with free space is not found for the row, the row is inserted on the last page, or if no space is available on the last page a new page is created and the row is inserted on the new page. New rows are then appended until some predefined amount of space is filled before a search is done again. As a result, insertion of a row into the database management system does not incur the worst-case cost of searching the entire free space map.

METHOD FOR EFFICIENTLY SEARCHING FREE SPACE IN A RELATIONAL DATABASE

FIELD OF THE INVENTION

The present invention relates to database management systems and more particularly to a method for searching free space in a relational database management system.

BACKGROUND OF THE INVENTION

A database management system (DBMS) comprises the combination of an appropriate computer, direct access storage devices (DASD) or disk drives, and database management software. A relational database management system is a DBMS which uses relational techniques for storing and retrieving information. The relational database management system or RDBMS comprises computerized information storage and retrieval systems in which data is stored on disk drives or DASD for semi-permanent storage. The data is stored in the form of tables which comprise rows and columns. Each row or tuple has one or more columns.

When inserting a row into a table in a relational database management system (RDBMS), a search for free space must be done to find space for the row being inserted. A common method utilized in the art to keep track of free space is the use of a free space map. A free space map comprises a series of space map pages (SMP's). Each space map page comprises an array of free space entries for a series of pages which form part of the table. Each entry in the space map page indicates the approximate amount of free space available in the page indexed in the array. When a row is to be inserted, the free space map is searched for a page with sufficient

free space. The search usually starts at the beginning of the free space map or from the position where space was last found. In either case, the entire free space map is searched until a page with space is found or until the free space map has been searched. If a page with free space is found, the row is inserted onto the page. If no free space is found, the row is appended to the table.

Using this searching technique, if the table grows to a very large size, there will likely be a proportionate increase in the number of space map pages in the free space map. As a result, it is possible that a page with sufficient free space for the row will be some distance from the starting position of the search resulting in the majority of the space map pages being searched. Because of the number of space map pages being searched, the performance during an insert operation will typically be compromised. In addition, each insert operation can incur the cost of searching the entire free space map without finding any space.

Accordingly, there remains a need for a method for efficiently searching free space in a database management system (DBMS) or a relational database management system (RDBMS).

SUMMARY OF THE INVENTION

The present invention provides a method and system for efficiently searching space map pages (SMP's) in a free space map in a database management system (DBMS) or a relational database management system (RDBMS). The method according to the invention limits the search to some number of space map pages in the free space map. Advantageously, a row insert operation does not need to incur the worst-case cost of searching the entire free space map.

According to the invention, the number of space map pages (SMP's) searched in a free space map for a database management

system is limited when a row is to be inserted in a table in the database. If the configured number of space map pages in the free space map are searched and a page with free space is not found for the row, the row is inserted on the last page or if no space is available on the last page a new page is created and the row is inserted on the new page. New rows are then appended until some amount of space (e.g. a predefined number of pages) is filled before a search is done again. Subsequent searches are started from where the previous search ended, regardless of whether a page with free space was found. The searching continues in a circular fashion. If consecutive searches fail to find free space, then the amount of space to be filled by new data before searching again is increased for each failed search. The user is allowed to configure the maximum amount of space to append to the table before searching is resumed.

If no free space is found after an exhaustive search of the free space map, the table is put in a Table Full State. While the table is in a Table Full State, a search is not done. Instead, all new rows of data are appended to the table. The Table Full State is reset when a specified amount of space is freed by deleting rows from the table or updating rows in the table which cause the rows to shrink in size. The amount of space to be freed to reset the Table Full State flag is either set to a default value or by the user.

In one aspect, the present invention provides a method for searching a table for free space for inserting data into the table in a database management system, wherein the table resides in a storage media and comprises a series of pages capable of storing data, and wherein the availability of free space in the table pages is indicated by a free space map having a plurality of space map

pages and each space map page including a free space indicator for each table page indexed in the space map page, the method comprises the steps: (a) limiting the number of space map pages to be searched for free space to a predetermined number; (b) searching the free space indicator in each of the predetermined number of space map pages to determine if any free space is available to insert the data; (c) if available free space is found, inserting the data into the table page containing the free space; (d) if sufficient free space is not found in any of the searched space map pages, appending data to the table until a predetermined amount of space has been added to the table; and (e) resuming searching of the space map pages for free space for inserting new data after the predetermined amount of space has been filled with data.

In another aspect, the present invention provides a relational database management system for use with a computer system wherein transactions are entered for inserting rows of data into a table contained in storage media in the computer system, wherein the table comprises a series of pages capable of storing the rows of data, and wherein the availability of free space in the table is indicated by a free space map having a plurality of space map pages with each space map page including a free space indicator for each page in the table indexed by the space map page, the system comprises: (a) means for processing a transaction to insert a row of data in the table; (b) means for locating free space in the table for inserting the row of data in response to a transaction to insert a row of data; (c) means for searching the space map pages in the free space map to locate sufficient free space for inserting the row of data; (d) the means for searching the space map pages includes, (i) means for limiting the number of space map pages to be searched in response to a data insert transaction; (ii) means

for inserting the row of data into the table if sufficient available free space is located; (iii) means for appending data to the table until a predetermined amount of space has been added to the table if sufficient free space is not found in any of the searched space map pages; and (iv) means for resuming searching of the space map pages for free space for inserting new data after the predetermined amount of space has been filled with data.

In yet another aspect, the present invention provides a computer program product for use on a computer system wherein transactions are executed for inserting data into a table in a database, the computer program product comprises: a recording medium; means recorded on the medium for instructing said computer system to perform the steps of: (a) limiting the number of space map pages to be searched for free space to a predetermined number; (b) searching the free space indicator in each of the predetermined number of space map pages to determine if any free space is available to insert the data; (c) if available free space is found, inserting the data into the table page containing the free space; (d) if sufficient free space is not found in any of the searched space map pages, appending data to the table until a predetermined amount of space has been added to the table; and (e) resuming searching of the space map pages for free space for inserting new data after the predetermined amount of space has been filled with data.

BRIEF DESCRIPTION OF THE DRAWINGS

Reference will now be made to the accompanying drawings which show, by way of example, preferred embodiments of the present invention, and in which:

Figs. 1(i) to 1(v) show a method according to the present

invention for efficiently searching free space in a relational database management system (RDBMS); and

Fig. 2 shows a method utilized by the method of Fig. 1 to account for data updates and deletions to the table.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention provides a method or process for efficiently searching free space in a relational database management system (RDBMS) in response to a request to insert data. While the method 100 according to the present invention is described in the context of a database management system, it is to be understood that the invention has wider applicability to other data storage systems where free space needs to be found in order to save data.

In a database management system (DBMS), data is stored in the form of tables which comprise rows and columns. Each row, also known as a tuple, has one or more columns. When a new row is to be inserted into a table, a search must be made to find space for the new row of data being inserted. Typically, a table comprises a series of pages, which are stored on a Direct Access Storage Device (DASD) or a hard disk. A free space map is utilized to keep track of the free space in the pages in the storage media. A free space map comprises one or more space map pages or SMP's. Each space map page comprises an array of elements where each element corresponds to a free space estimate for a respective page in the database management system. The page number provides an index for the array and typically the array indexes (i.e. pages) are arranged sequentially, for example, the first space map page or SMP holds free space entries for pages 1 to 500, the second SMP holds free space entries for pages 501 to 1000, and so on.

As will now be described, the method according to the present invention allows the user to limit the number of free space map pages that are searched for each row inserted and to define some amount of space (e.g. a predefined number of pages) for appending data before a search is done again. The method according to the invention is incorporated into the computer program comprising the database management system in a process or thread, for example, which is invoked to insert a row of data into the table in the database management system. If the configured number of space map pages have been searched without finding a page for the row, the row will be appended to the table. The predefined number of new pages will be filled before another search is done. Subsequent searches start from where the previous search ended whether or not a page with free space was found. The search continues in a circular fashion, that is, once the end of the free space map is reached, the search resumes at the beginning of the free space map. If consecutive searches fail, for each failed search, the predefined number of pages to fill before searching again is increased. The user can configure the maximum value for the predefined number of pages for appending data before searching will resume.

Reference is now made to Fig. 1(i) which shows a method or process 100 for efficiently searching a free space map such as found in a DBMS or RDBMS. The method or process 100 is executed by a process or thread for inserting data into a page in a table in the database. As will now be described, the method 100 searches the free space map for the table to find a page on which there is sufficient free space to insert the data, i.e. a row or tuple.

As shown in Fig. 1(i), the first step comprises a decision block 102 which checks two flags, SEARCH and FREE_SPACE. The flags

SEARCH and FREE_SPACE, and other flags and variables, described below comprise global cached variables with initial values which are updated by the process or thread that inserts the data while the variables SMPS_SEARCHED, FIRST_PAGE_SEARCHED and SEARCH_ENTIRE_TABLE are local variables which are maintained by the process/thread that inserts the data. The SEARCH flag indicates whether the free space map is to be searched, and the FREE_SPACE flag indicates whether there is free space available in the table. If the SEARCH flag is FALSE, indicating that a search of the free space map is not to be made, or if the FREE_SPACE flag is FALSE, indicating that there is no free space available in the table, then processing moves to decision block 103 in Fig. 1(iv). As will be described in further detail below, the FREE_SPACE flag and the SEARCH flag will be FALSE if the table is put into a "Table Full State". The SEARCH flag is also set to FALSE when a search for free space fails to find space as will also be described in further detail below.

On the other hand, if the table is not full and a search is to be done, then a variable SMPS_SEARCHED is reset in block 104. Also in block 104, a variable FIRST_PAGE_SEARCHED is set to the CURRENT_PAGE indexed in the table. The SMPS_SEARCHED variable indicates the number of space map pages (SMP's) which have been searched in the free space map, and the variable FIRST_PAGE_SEARCHED indicates the first page in the table which will be checked for free space to insert this row of data. Referring still to block 104 in Fig. 1(i), a variable CURRENT_SMP is set to the space map page containing the entry for the page indicated by the variable CURRENT_PAGE. Lastly in block 104, a flag SEARCH_ENTIRE_TABLE is set to FALSE. The SEARCH_ENTIRE_TABLE flag indicates whether the entire table is to be searched.

Next in decision block 106, the space map page indicated by the variable CURRENT_SMP is checked to determine if the current space map page indicates that the current page in the table (i.e. CURRENT_PAGE) has sufficient free space for the row to be inserted. If the space map page indicates sufficient free space, then an actual check of the current page is made in decision block 108 to determine if the free space indicated in the space map page is useable. If sufficient useable free space is found in the current page of the table, then the flag SEARCH is set to TRUE in block 110, and another flag SPACE_FOUND is also set to TRUE. The SPACE_FOUND flag indicates that sufficient free space was found to insert the data, while the SEARCH flag is set to indicate that a successful search was conducted. In block 110, a variable SEARCH_START_PAGE is set to the current page (i.e. CURRENT_PAGE), and variables TIMES_TABLE_SEARCHED, AMOUNT_TO_APPEND and AMOUNT_APPENDED are also set to ZERO. The variable SEARCH_START_PAGE is set to the current page so that subsequent insert operations know on which page space was last found. It will be understood the SEARCH_START_PAGE variable is a global variable which indicates the page number of the first page searched since space was last found by any insert operation, and the FIRST_PAGE_SEARCHED variable is a local variable which indicates the page number of the first page searched for the current insert. The TIMES_TABLE_SEARCHED variable indicates the number of times the entire free space map for the table has been searched without finding free space. As will be described in further detail below, preferably no more than two searches of the entire free space map are allowed before the table is put into a Table Full State. For a table with fixed length columns only, the table is placed in the Table Full State after the entire free space map has been searched

once without finding free space. In this case, free space was found so the TIMES_TABLE_SEARCHED variable is set to ZERO. The AMOUNT_TO_APPEND variable provides a running total of the amount of space to append before the free space map is to be searched again. If consecutive searches of the free space map fail to find free space, for each failed search, the amount of space is increased by some increment to a maximum value as described in further detail below in Fig. 1(v). The maximum amount of space to be appended before searching is resumed can be configured by the user. The AMOUNT_APPENDED variable indicates the amount of space which has been appended since the start of the appending operation.

It will be appreciated that maintaining a Table Full State eliminates the search overhead when it is known that there is no free space available in the table. Furthermore, allowing the user to configure the amount of space to be freed before resetting the Table Full State provides additional control in trading off insert performance versus space reuse as will be described in more detail below.

Having found free space in the current page, the data is inserted into the page in block 112. The searching and row insertion procedure is done and processing then returns to the calling thread or process in block 114.

If the entry in the space map page indicates that space is not available on the current page (decision block 106), then the CURRENT_PAGE variable is incremented in block 116 in order to search for free space on the next page in the table. In decision block 118, a check is made to determine if the new current page is past the last page in the table, by comparing the CURRENT_PAGE variable to a variable LAST_PAGE. The LAST_PAGE variable holds the page number of the last page in the table. If the page number in

CURRENT_PAGE does not exceed the LAST_PAGE number, then a check is made to determine if the current page is on the next space map page (decision block 120). If the current page is located on the next space map page, then the CURRENT_SMP variable is advanced to the next space map page in block 122. Referring back to decision block 118, if the CURRENT_PAGE number exceeds the LAST_PAGE number, then the search is set to commence back at the beginning of the free space map and the start of the table. The CURRENT_SMP variable is set to ZERO and the CURRENT_PAGE variable is also set to ZERO (block 124).

Next in decision block 126 (Fig. 1(ii)), a check is made to determine if more than 50% of the entries in the previous space map page have been searched. If more than 50% of the entries have been searched, then the space map page is included in the number of space map pages searched and the SMPS_SEARCHED variable is incremented (block 128). Once the value for the SMPS_SEARCHED variable reaches the maximum value defined by the SEARCH_LIMIT variable, further searching of the space map pages stops and the row is appended to the end of the table. If less than 50% of the entries in the space map page have been searched, then the SMPS_SEARCHED variable is not incremented.

Referring still to Fig. 1(ii), a check is made in decision block 130 to determine if the SEARCH_START_PAGE is the same as the CURRENT_PAGE. If TRUE, this means that the search has wrapped around and the free map space for the entire table has been searched without finding space. A further check is made of flag SEARCH_ENTIRE_TABLE in decision box 132. The SEARCH_ENTIRE_TABLE flag indicates whether the entire table is to be searched for the current insert. If the SEARCH_ENTIRE_TABLE flag is FALSE, then the entire table is not to be searched and a variable

TIMES_TABLE_SEARCHED is incremented in block 133 (Fig. 1(iii)) as will be described in more detail below. The TIMES_TABLE_SEARCHED variable indicates the number of the times the table has been searched without finding free space to insert one or more rows of data. On the other hand, if the SEARCH_ENTIRE_TABLE flag is set to TRUE, then the entire table has to be searched for the current insert, and a check is made at decision box 134 to determine if the FIRST_PAGE_SEARCHED is the same as the CURRENT_PAGE. If TRUE, then the entire table has been searched for the current insert and a further check is made at decision block 135 in Fig. 1(v) as will be described in more detail below.

Referring again to Fig. 1(ii), if the FIRST_PAGE_SEARCHED is not the same as the CURRENT_PAGE, then a check of the SEARCH_ENTIRE_TABLE flag is made in decision block 136. If TRUE, then the entire table is to be searched, and a check is made in decision block 106 (Fig. 1(i)) to determine if the current space map page indicates free space on the current page. If the SEARCH_ENTIRE_TABLE flag is FALSE, i.e. the entire table is not to be searched, then a check is made in decision block 138 to determine if the number of space map pages searched is equal to the maximum number of space map pages specified in the variable SEARCH_LIMIT. If the maximum number of space map pages have not been searched, then searching continues and the CURRENT_SMP is checked for free space in decision block 106 (Fig. 1(i)) as described above. On the other hand, if the SEARCH_LIMIT for SMPS_SEARCHED has been reached then a check is made of flag SPACE_FOUND in decision block 139 in Fig. 1(v) as will be described in more detail below.

Reference is next made to Fig. 1(iv). If the SEARCH flag is FALSE and the FREE_SPACE flag is FALSE (decision block 102 in Fig.

1(i)), then there is no free space in the table and further searching is not to be done. As will now be explained, the method 100 appends the row to the table by first attempting to insert the row of new data on the last page in the table or adding a new page to the table if there is no space available on the last page. As shown in Fig. 1(iv), a check is made in decision block 103 to determine if there is space available on the last page. If yes, then the data is inserted on the last page in the table (block 140) and processing returns to the calling process/thread (block 142).

If there is not enough space available on the LAST_PAGE (decision block 103), then a further check is made in decision block 144 of the FREE_SPACE flag. The FREE_SPACE flag indicates whether there is free space available in the table. If there is no free space available, i.e. the FREE_SPACE flag is FALSE, then a check is made in decision block 146 to determine if the storage media (e.g. DASD or hard disk) is full. If the storage media is full, then an error has occurred (block 148) and processing returns to the calling process/thread (block 150). On the other hand, if there is still space available in the storage media, a new page for the table is created (block 152) and this becomes the new last page in the table and the LAST_PAGE variable is incremented by one (block 152). The row of data is inserted on the new page (block 154) and processing returns to the calling process/thread (block 156).

Referring to Fig. 1(iv), if there is free space available in the object (decision block 144), then a check is made in decision block 158 to determine if the variable AMOUNT_APPENDED is greater than or equal to the variable AMOUNT_TO_APPEND. The AMOUNT_APPENDED variable indicates the amount of space appended since the start of appending data to the table. The AMOUNT_TO_APPEND variable provides

the total amount of space to append before another search is made for space in the table. The AMOUNT_TO_APPEND variable is incremented by a value *m* each time consecutive searches fail to find space, subject to a maximum amount specified by *max_amount*, where *max_amount* is either specified by the user or a default value from the system. If the AMOUNT_APPENDED is less than the AMOUNT_TO_APPEND, then a check is made to determine if the storage media is full (block 160). If the storage media is not FULL, then there is space to create a new page which is appended to the end of the table. In block 162, a new page is created and the LAST_PAGE variable is incremented by ONE. The AMOUNT_APPENDED variable is also increased by the size of the page appended (PAGE_SIZE). After the new page is created, the row of data is inserted on the new page (block 164) and processing returns to the calling process/thread (block 166).

Referring still to Fig. 1(iv), if the AMOUNT_APPENDED is less than the AMOUNT_TO_APPEND (decision box 158) but the storage media is full (decision box 160), then any free space must be found by searching the free space map. A search of the space map pages in the free space map is commenced starting at the current page in the table. As shown in block 168, the SEARCH_ENTIRE_TABLE flag is set TRUE, the variable SMPS_SEARCHED is set to ZERO, the variable FIRST_PAGE_SEARCHED is set to the current page in the table (i.e. variable CURRENT_PAGE) and the variable CURRENT_SMP is set to the space map page with an entry for the current page in the table. Searching resumes with a check in decision block 106 (Fig. 1(i)) to determine if the current space map page indicates free space on the current page.

Referring still to Fig. 1(iv), if the AMOUNT_APPENDED is greater than or equal to the AMOUNT_TO_APPEND, then the limit for

appending pages has been reached and any free space must be located by searching the free space map. The search is resumed by first performing the following operations in block 170: setting the SEARCH flag to TRUE, clearing the TIMES_TABLE_SEARCHED variable and the AMOUNT_APPENDED variable, setting the SEARCH_ENTIRE_TABLE flag to FALSE, setting the FIRST_PAGE_SEARCHED variable to the current page in the table (i.e. variable CURRENT_PAGE), and setting the variable CURRENT_SMP to the space map page with an entry for the current page in the table. Searching then resumes with a check in decision block 106 (Fig. 1(i)) to determine if the current space map page indicates free space on the current page.

It will be appreciated that the first page in a search that did not find space is cached if the previous search found space. The cached page is initialized to the first page in the table before any search is performed. This cached page is used to determine when the entire free space map has been searched. While the table is in a Table Full State, a search is not done. Instead, all rows will be appended to the table (blocks 152-154 in Fig. 1(iv)). The table full state is reset when some amount of space is freed from the table by deletion of rows or by updates to rows which cause the rows to shrink in size as will be described below with reference to Fig. 2.

As described above with reference to Fig. 1(ii), if the flag SEARCH_ENTIRE_TABLE is FALSE, i.e. the entire table is not to be searched, then the variable TIMES_TABLE_SEARCHED is incremented in block 133 (Fig. 1(iii)). Referring to Fig. 1(iii), after the TIMES_TABLE_SEARCHED variable is incremented, a check is made in decision block 172 to determine if the data for the table comprises fixed length columns only. If the data is not fixed length, a check is made in decision box 174 to determine if the table (i.e.

the free space map) has been searched two times. For tables with variable length columns, the entire map will be searched twice for an exhaustive search. If no free space is found after an exhaustive search of the space map pages in the free space map, the table is put in a "Table Full State". If the table has only been searched once, then the search resumes by first checking decision box 106 as described above with reference to Fig. 1(i). If the table has been searched twice without finding space, then the table is placed in the "Table Full State" as indicated in box 176. The FREE_SPACE flag is set to FALSE, the SPACE_FREED variable is set to ZERO (i.e. amount of space freed since the FREE_SPACE flag was set to FALSE), and the variable FIRST_PAGE_WITH_SPACE is set to NEGATIVE ONE. The variable FIRST_PAGE_WITH_SPACE indicates the page closest to the start of the table with free space, and is set to the smallest page number (i.e. closest to the beginning) in the table from which space was freed after the flag FREE_SPACE is set to FALSE. In the case of a table with fixed length columns only (i.e. decision bock 172 is TRUE), the table is placed in the Table Full State in block 176 after the entire free space map has been searched once for an exhaustive search. Next in decision block 178, a check is made to determine if the storage media for the table (e.g. disk drive memory or DASD) is full. If the media is full, an error is registered (block 180) and further searching is terminated and control returns to the calling process/thread (block 182). On the other hand, if there is still space in the storage media (i.e. decision box 178 is FALSE), then a new page is created and appended to the table (block 184) and the last page number in the table (i.e. variable LAST_PAGE) is incremented by ONE, and the row of data is inserted on this new page appended to the table (block 186), and control returns to the calling process/thread (block

182).

It will be appreciated that the reason for searching the free space map twice for tables with variable length columns is that later rows being inserted could be smaller than earlier rows. While there may not have been enough space for the larger rows, the smaller rows may fit in the free space available in the table. The exhaustive search will be done over a number of inserts where each search does not find free space unless the free space map contains a smaller number of pages than the value configured for the maximum number of free space map pages to search.

Referring back to Fig. 1(ii), if the first page checked for space for the current row insert request (i.e. variable FIRST_PAGE_SEARCHED) is the same as the current page to be searched for space (decision block 134 in Fig. 1(ii)), then a check is made to determine if the entire table is to be searched (i.e. the SEARCH_ENTIRE_TABLE flag) in decision block 135 in Fig. 1(v). If the entire table is to be searched, then a check is made to determine if the storage media is full in decision block 188. If the storage media is full, then an error is registered (block 190) and the searching process and control returns to the calling process/thread(block 192). On the other hand, if there is still space in the storage media, the following operations are performed in block 194: a new page is appended to the table, the LAST_PAGE variable is incremented by ONE to indicate the new last page in the table, and the AMOUNT_APPENDED variable is increased by the page size of the new page. Next the row of data is inserted on the new page (block 196) and the searching procedure is completed and control returns to the calling process/thread (block 198).

If the maximum number of space map pages have been searched, i.e. SMPS_SEARCHED equals SEARCH_LIMIT (decision block 138 in Fig.

1(ii)), then the SPACE_FOUND flag is checked in decision block 139 in Fig. 1(v). The SPACE_FOUND flag indicates whether any free space was found during the previous search of the space map pages in the free space map. If no free space was found, i.e. SPACE_FOUND flag is FALSE, then a check is made in decision block 200 to determine if the running total of the amount of space appended (i.e. AMOUNT_TO_APPEND) exceeds the maximum amount of space which can be appended. If the maximum amount has not been reached, the variable AMOUNT_TO_APPEND is incremented by a value *m* in block 202. The variable AMOUNT_TO_APPEND is incremented by *m* subject to the maximum MAX_AMOUNT each time consecutive searches fail to find space. The variable MAX_AMOUNT contains a value *x* which is a multiple of *m* and is either set by the user or takes a default value from the DBMS or RDBMS. If the maximum amount of space is already to be appended, then step 202 is bypassed. On the other hand, if space was found for the previous search (i.e. SPACE_FOUND is TRUE), then the SPACE_FOUND flag is set to FALSE, the SEARCH flag is set to FALSE because space was not found and the rows will be appended to the table, and the variable AMOUNT_TO_APPEND is set to the value *m* in block 203.

Referring still to Fig. 1(v), next a check is made in decision box 204 to determine if there is sufficient space available on the LAST_PAGE in the table for the row. If there is space on the last page in the table, the row of data is inserted (block 206) and the searching and row insertion process is completed and control returns to the calling process/thread (block 208). If there is no space available on the last page, then a check is made in decision box 210 to determine if the storage media is full. If TRUE, then additional disk space cannot be allocated to store the row, and an exhaustive search of the entire free space map will be done before

an error is returned. Accordingly in block 212, the flag SEARCH_ENTIRE_TABLE is set to TRUE to indicate that the entire table (i.e. free space map for the table) is to be searched, and searching resumes on the current SMP (decision block 106 in Fig. 1(i)). If there is still space in the storage media (decision block 210), then the following operations are performed in block 214: a new page is appended to the table, the LAST_PAGE variable is incremented by ONE to indicate the new last page in the table, and the AMOUNT_APPENDED variable is increased by the page size of the new page. Next the row of data is inserted on the new page (block 216) and the free space searching and row insertion procedure is completed and control returns to the calling process/thread (block 218).

As described above, the table is put into a Table Full State if no free space is found after an exhaustive search of the free space map. While in the Table Full State no searching is done and all new rows of data are appended to the table. The Table Full State is reset when some amount of space is freed as a result of rows being deleted from the table or updates being made to rows which cause the rows to shrink in size as will now be described with reference to Fig. 2.

Reference is made to Fig. 2 which shows a method, indicated generally by reference 222, for accounting for space freed in the table. The method 222 is invoked in response to a command to delete a row of data or a command to update a row of data which results in the freeing of some space in the table (block 224). After execution of the delete or update command by the process/thread (block 226), the FREE_SPACE flag is checked in decision box 228 to determine if there is any available free space in the table. If there is still available free space, independent of the deletion or update,

control is returned to the calling process/thread (block 229).

If the FREE_SPACE flag was set to FALSE, i.e. indicating that there is no available free space in the table, then the amount of space freed through the previous delete or update operation is added to the variable SPACE_FREED (block 230). The SPACE_FREED value represents the amount of space freed since the FREE_SPACE flag was set to FALSE. Next in decision block 232, a check is made to determine if the variable FIRST_PAGE_WITH_SPACE is set to NEGATIVE ONE. As described above, the FIRST_PAGE_WITH_SPACE variable is set to negative one when a Table Full State is reached (block 176 in Fig. 1(iii)) and indicates that there are no pages in the table with free space. If FIRST_PAGE_WITH_SPACE contains a page number, that page is compared with the page from which the space was freed (decision block 234). If there is no page with free space (i.e. FIRST_PAGE_WITH_SPACE = -1) or the page from which space was freed is earlier in the table compared to the page number stored in the variable FIRST_PAGE_WITH_SPACE, then the variable FIRST_PAGE_WITH_SPACE is set to this page (block 236). Next, a check is made in decision block 238 to determine if the SPACE_FREED since the FREE_SPACE flag was set to FALSE is equal to or exceeds the amount specified in variable SPACE_TO_FREE. The SPACE_TO_FREE variable specifies the amount of space *y* that needs to be freed before the FREE_SPACE flag can be reset and searching of the free space map resumed. The value *y* is either set by the user or as a default value by the DBMS or RDBMS. If the amount of space freed is greater than or equal to the value in SPACE_TO_FREE, then the Table Full State is reset and searching is enabled in block 240 as follows: the FREE_SPACE flag is set to TRUE indicating that free space is available in the object, the SEARCH flag is set to TRUE indicating that a search for free space can be made, the

SPACE_FOUND flag is set to TRUE, the CURRENT_PAGE variable is set to the FIRST_PAGE_WITH_SPACE, the SEARCH_START_PAGE is set to the CURRENT_PAGE, and the variable TIMES_TABLE_SEARCHED is set to ZERO, and control returns (block 229).

It will be appreciated that the method according to the present invention limits the number of free space map pages which are searched in response to a request to insert a row of data. The insertion of a new row does not incur the cost of searching the entire free space map. Since rows are appended until the predefined number of pages are filled, there will be some delay before the next search is performed. When consecutive searches fail to find space, i.e., it appears unlikely that a page with sufficient free space will be found, the predefined number of pages to append before a search is done again is incremented for each failed search until the maximum configured amount is reached. This allows more activity on the table that could potentially free up space, thereby increasing the probability that a future search will find space. The overhead of searching for free space is eliminated for most rows inserted when it appears unlikely that free space will be found. Since searching will resume once the number of pages to be appended have been filled, free space will eventually be reused. Furthermore allowing the user to configure both the number of free space map pages to search and the maximum number of pages to append before searching resumes provides the user with a mechanism to trade-off row insertion performance with space reuse. In the case where row insertion performance is more important than space reuse, the user would configure the search distance to be small and the number of pages to append to be large. In the case where space reuse is more important, the user would configure the number of pages to append to be small and the search distance to be

large.

The present invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. Therefore, the presently discussed embodiments are considered to be illustrative and not restrictive, the scope of the invention being indicated by the appended claims rather than the foregoing description, and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced therein.

The present invention is not limited to the specifically disclosed embodiments, and variations and modifications may be made without departing from the scope of the present invention.

CLAIMS

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A method for searching a table for free space for inserting data into the table in a database management system, wherein the table resides in a storage media and comprises a series of pages capable of storing data, and wherein the availability of free space in the table pages is indicated by a free space map having a plurality of space map pages and each space map page including a free space indicator for each table page indexed in the space map page, said method comprising the steps:

(a) limiting the number of space map pages to be searched for free space to a predetermined number;

(b) searching the free space indicator in each of said predetermined number of space map pages to determine if any free space is available to insert the data;

(c) if available free space is found, inserting the data into the table page containing the free space;

(d) if sufficient free space is not found in any of the searched space map pages, appending data to the table until a predetermined amount of space has been added to the table; and

(e) resuming searching of the space map pages for free space for inserting new data after the predetermined amount of space has been filled with data.

2. The method as claimed in claim 1, further including the step of increasing the predetermined amount of space each time a search for sufficient free space to insert new data fails.

3. The method as claimed in claim 1 or 2, further including the steps of placing the table in a full state if sufficient free space is not found after a search of the entire free space map for the table, wherein further searching is curtailed until the full state is reset.

4. The method as claimed in claim 3, wherein the full state is reset in response to data being removed from the table, said removal resulting in sufficient free space for a subsequent insertion of data.

5. The method as claimed in claim 4, wherein the free space map is searched twice if the table comprises variable length columns, and wherein the free space map is searched once if the table comprises fixed length columns.

6. The method as claimed in claim 1, wherein said step of resuming searching comprises resuming searching on the space map page at which the previous search ended.

7. The method as claimed in claim 1, wherein a default value for the predetermined number of space map pages is provided by the database management system and said default value is changeable by a user.

8. The method as claimed in claim 1, wherein a default value for the predetermined amount of space is provided by the database management system and said default value is changeable by a user.

9. A relational database management system for use with a

computer system wherein transactions are entered for inserting rows of data into a table contained in storage media in the computer system, wherein the table comprises a series of pages capable of storing the rows of data, and wherein the availability of free space in the table is indicated by a free space map having a plurality of space map pages with each space map page including a free space indicator for each page in the table indexed by the space map page, said system comprising:

(a) means for processing a transaction to insert a row of data in the table;

(b) means for locating free space in the table for inserting the row of data in response to a transaction to insert a row of data;

(c) means for searching the space map pages in the free space map to locate sufficient free space for inserting the row of data;

(d) said means for searching the space map pages including,

(i) means for limiting the number of space map pages to be searched in response to a data insert transaction;

(ii) means for inserting the row of data into the table if sufficient available free space is located;

(iii) means for appending data to the table until a predetermined amount of space has been added to the table if sufficient free space is not found in any of the searched space map pages; and

(iv) means for resuming searching of the space map pages for free space for inserting new data after the predetermined amount of space has been filled with data.

10. A computer readable program product for use on a computer system wherein transactions are executed for inserting data into a

table in a database, wherein the table comprises a series of pages capable of storing data, and wherein the availability of free space in the table pages is indicated by a free space map having a plurality of space map pages and each space map page including a free space indicator for each table page indexed in the space map page, said computer program product comprising:

a recording medium;

means recorded on said medium for instructing said computer system to perform the steps of:

(a) limiting the number of space map pages to be searched for free space to a predetermined number;

(b) searching the free space indicator in each of said predetermined number of space map pages to determine if any free space is available to insert the data;

(c) if available free space is found, inserting the data into the table page containing the free space;

(d) , if sufficient free space is not found in any of the searched space map pages, appending data to the table until a predetermined amount of space has been added to the table; and

(e) resuming searching of the space map pages for free space for inserting new data after the predetermined amount of space has been filled with data.

11. An article comprising a computer readable data storage medium, means recorded on the medium to implement the method of any one of claims 1 to 8.

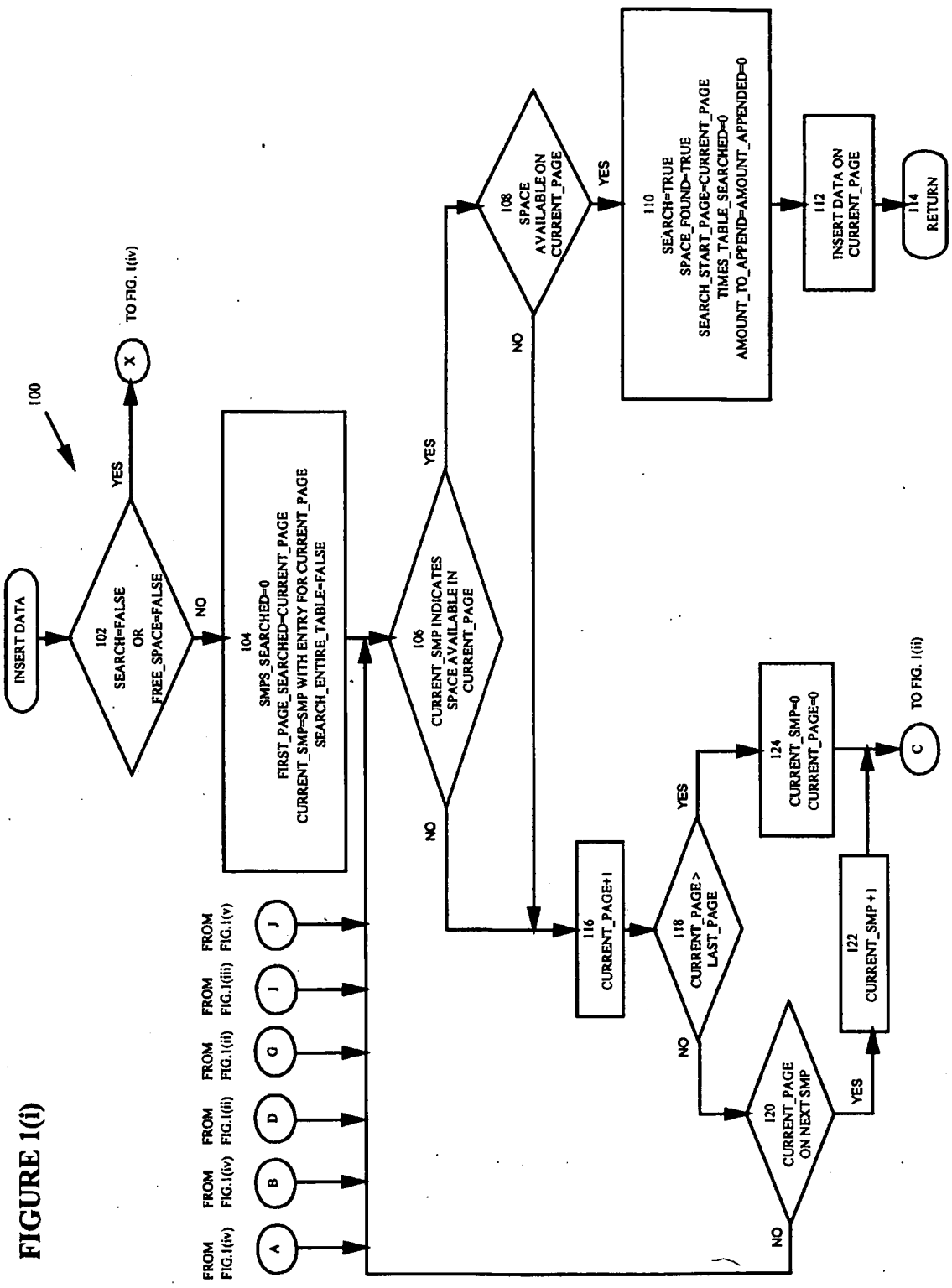

100
INSERT DATA
102
SEARCH=FALSE
OR
FREE_SPACE=FALSE
YES
X
TO FIG. 1(iv)
NO
104
SMPS_SEARCHED=0
FIRST_PAGE_SEARCHED=CURRENT_PAGE
CURRENT_SMP=SMP WITH ENTRY FOR CURRENT_PAGE
SEARCH_ENTIRE_TABLE=FALSE
106
CURRENT_SMP INDICATES
SPACE AVAILABLE IN
CURRENT_PAGE
YES
NO
108
SPACE
AVAILABLE ON
CURRENT_PAGE
YES
NO
110
SEARCH=TRUE
SPACE_FOUND=TRUE
SEARCH_START_PAGE=CURRENT_PAGE
TIMES_TABLE_SEARCHED=0
AMOUNT_TO_APPEND=AMOUNT_APPENDED=0
112
INSERT DATA ON
CURRENT_PAGE
114
RETURN
116
CURRENT_PAGE+1
118
CURRENT_PAGE >
LAST_PAGE
YES
NO
124
CURRENT_SMP=0
CURRENT_PAGE=0
C
TO FIG. 1(ii)
120
CURRENT_PAGE
ON NEXT SMP
YES
NO
122
CURRENT_SMP+1
A
FROM FIG.1(iv)
B
FROM FIG.1(iv)
D
FROM FIG.1(ii)
G
FROM FIG.1(ii)
I
FROM FIG.1(iii)
J
FROM FIG.1(v)

FIGURE 1(i)

FIGURE 1(ii)

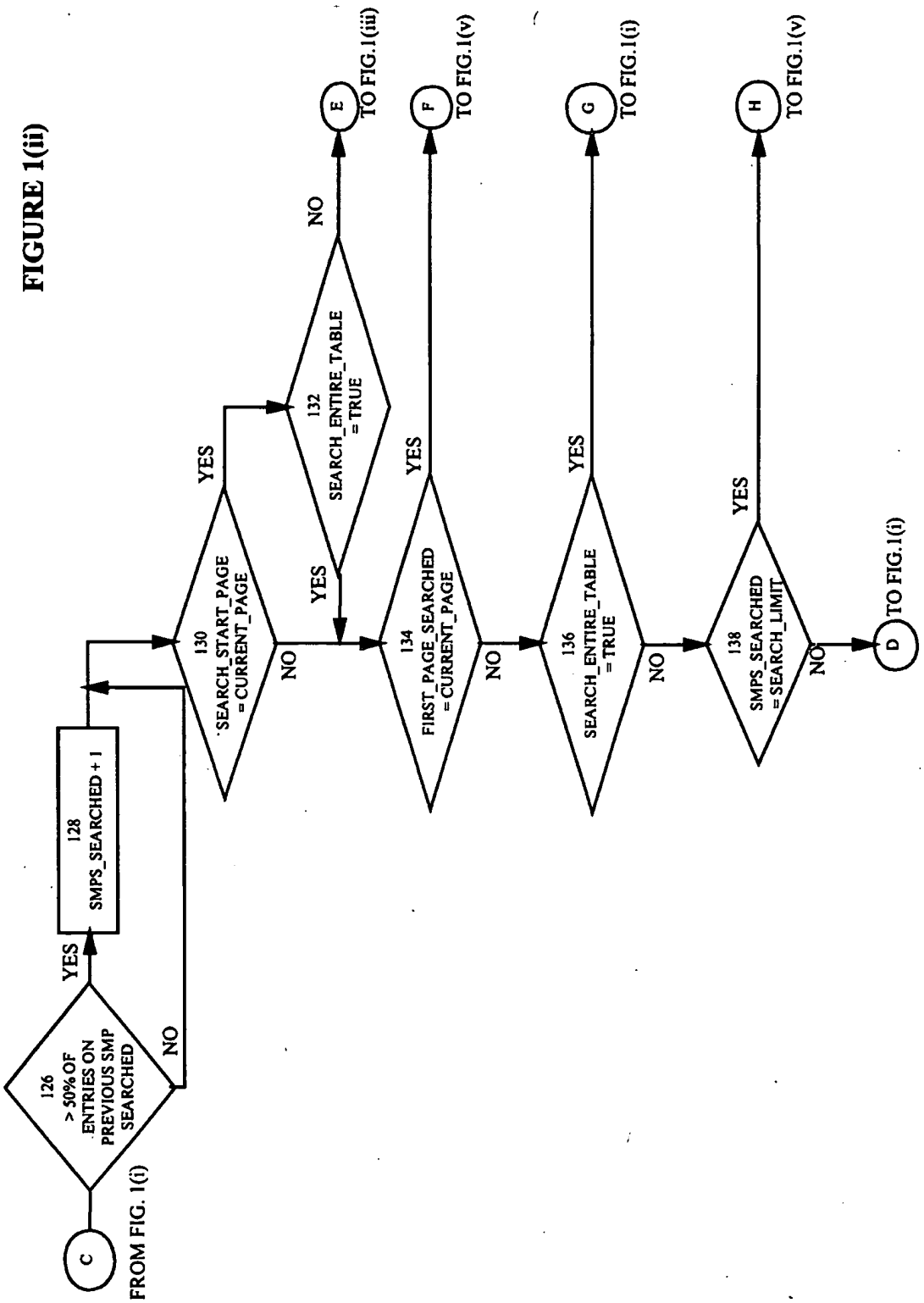

FROM FIG. 1(i)
C
126
> 50% OF ENTRIES ON PREVIOUS SMP SEARCHED
YES
NO
128
SMPS_SEARCHED + 1
130
SEARCH_START_PAGE = CURRENT_PAGE
YES
NO
132
SEARCH_ENTIRE_TABLE = TRUE
YES
NO
E
TO FIG.1(iii)
134
FIRST_PAGE_SEARCHED = CURRENT_PAGE
YES
NO
F
TO FIG.1(v)
136
SEARCH_ENTIRE_TABLE = TRUE
YES
NO
G
TO FIG.1(i)
138
SMPS_SEARCHED = SEARCH_LIMIT
YES
NO
H
TO FIG.1(v)
D
TO FIG.1(i)

FIGURE 1(iii)

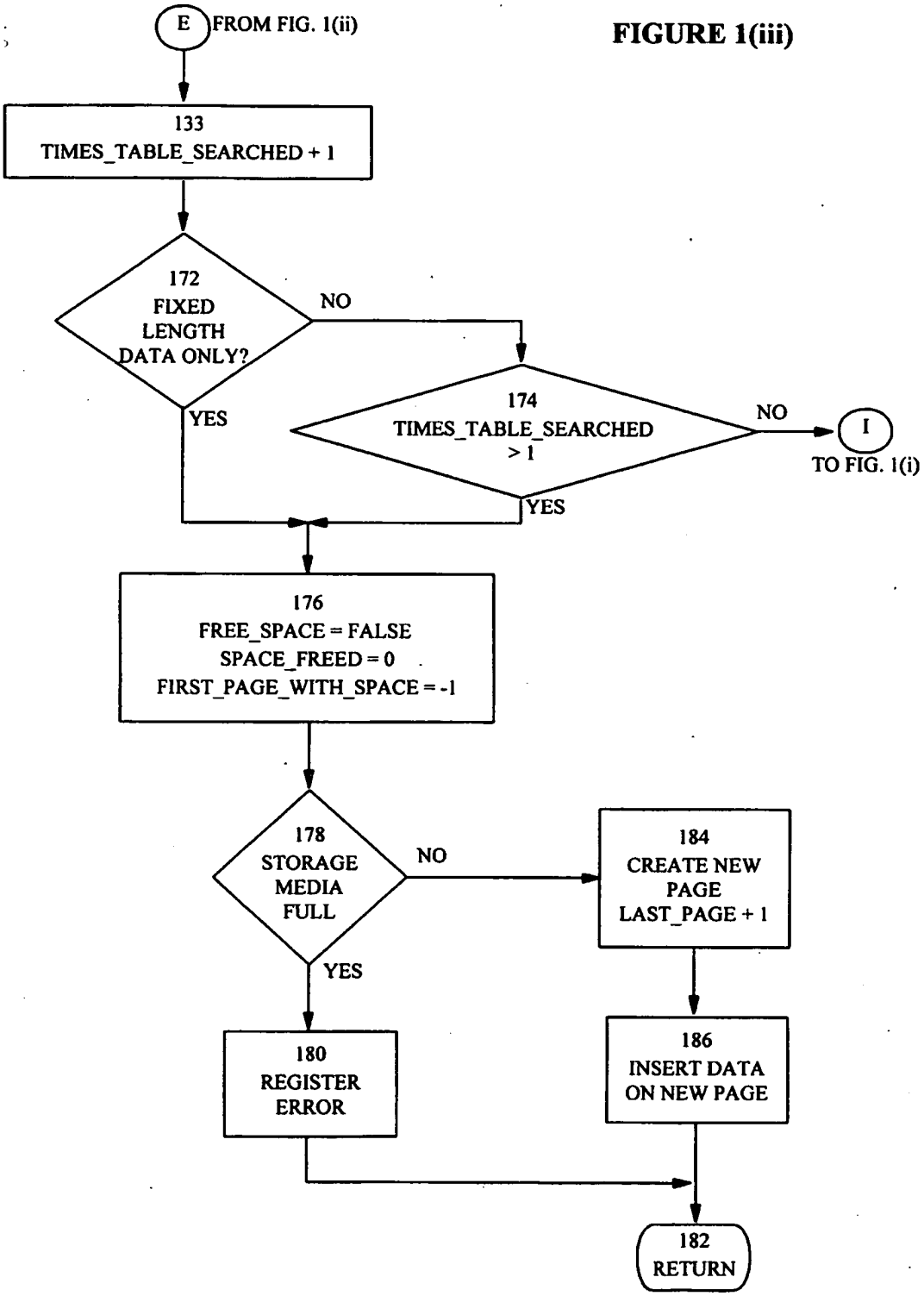

E
FROM FIG. 1(ii)
133
TIMES_TABLE_SEARCHED + 1
172
FIXED LENGTH DATA ONLY?
NO
YES
174
TIMES_TABLE_SEARCHED > 1
NO
I
TO FIG. 1(i)
YES
176
FREE_SPACE = FALSE
SPACE_FREED = 0
FIRST_PAGE_WITH_SPACE = -1
178
STORAGE MEDIA FULL
NO
184
CREATE NEW PAGE
LAST_PAGE + 1
YES
180
REGISTER ERROR
186
INSERT DATA ON NEW PAGE
182
RETURN

FIGURE 1(iv)

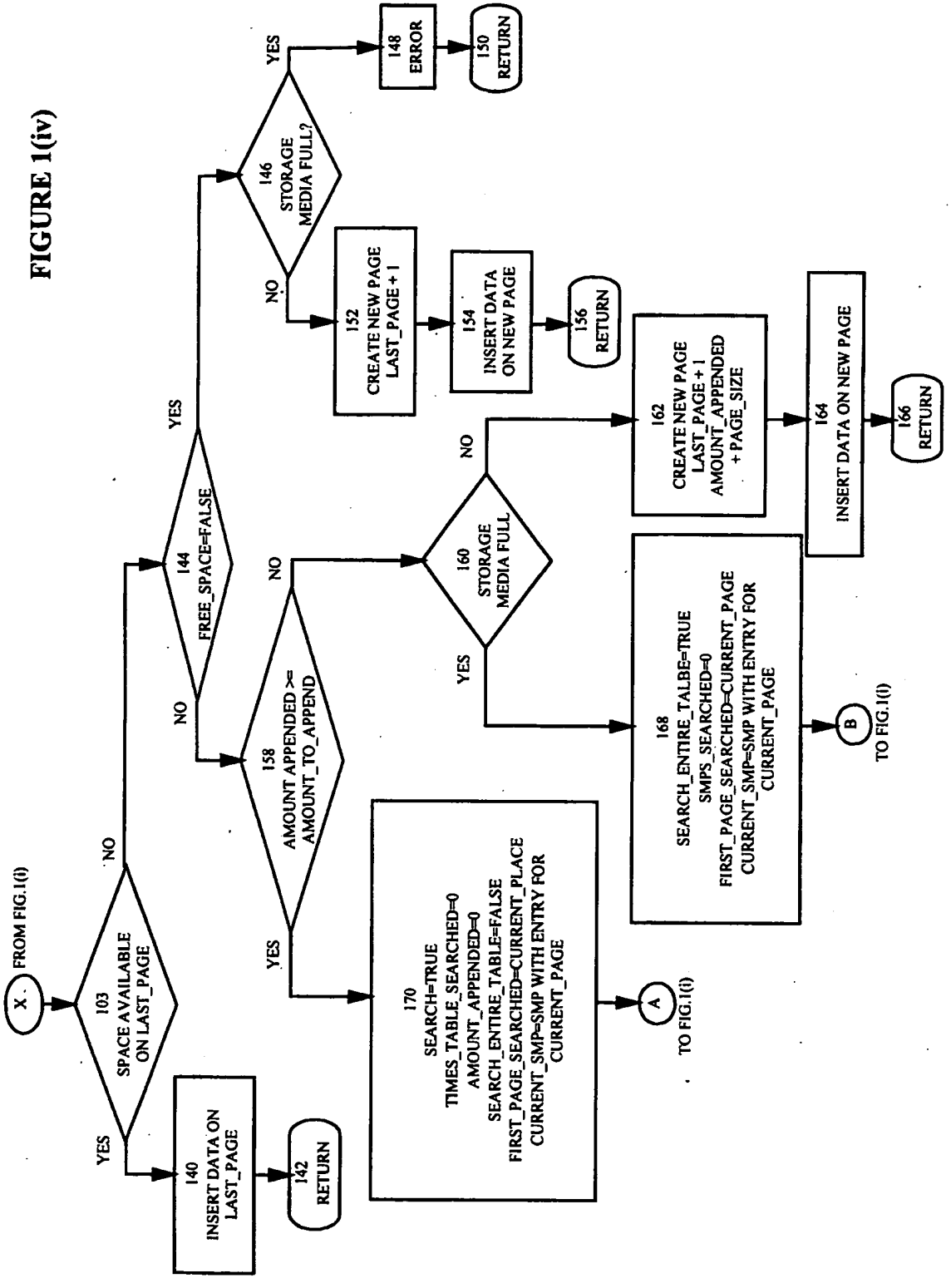

X
FROM FIG.1(I)
103
SPACE AVAILABLE ON LAST_PAGE
YES
NO
140
INSERT DATA ON LAST_PAGE
142
RETURN
144
FREE_SPACE=FALSE
YES
NO
146
STORAGE MEDIA FULL?
YES
148
ERROR
150
RETURN
NO
152
CREATE NEW PAGE LAST_PAGE + 1
154
INSERT DATA ON NEW PAGE
156
RETURN
158
AMOUNT APPENDED >= AMOUNT_TO_APPEND
NO
160
STORAGE MEDIA FULL
NO
162
CREATE NEW PAGE LAST_PAGE + 1 AMOUNT_APPENDED + PAGE_SIZE
164
INSERT DATA ON NEW PAGE
166
RETURN
YES
168
SEARCH_ENTIRE_TALBE=TRUE
SMPS_SEARCHED=0
FIRST_PAGE_SEARCHED=CURRENT_PAGE
CURRENT_SMP=SMP WITH ENTRY FOR CURRENT_PAGE
B
TO FIG.1(I)
YES
170
SEARCH=TRUE
TIMES_TABLE_SEARCHED=0
AMOUNT_APPENDED=0
SEARCH_ENTIRE_TABLE=FALSE
FIRST_PAGE_SEARCHED=CURRENT_PLACE
CURRENT_SMP=SMP WITH ENTRY FOR CURRENT_PAGE
A
TO FIG.1(I)

FIGURE 1(v)

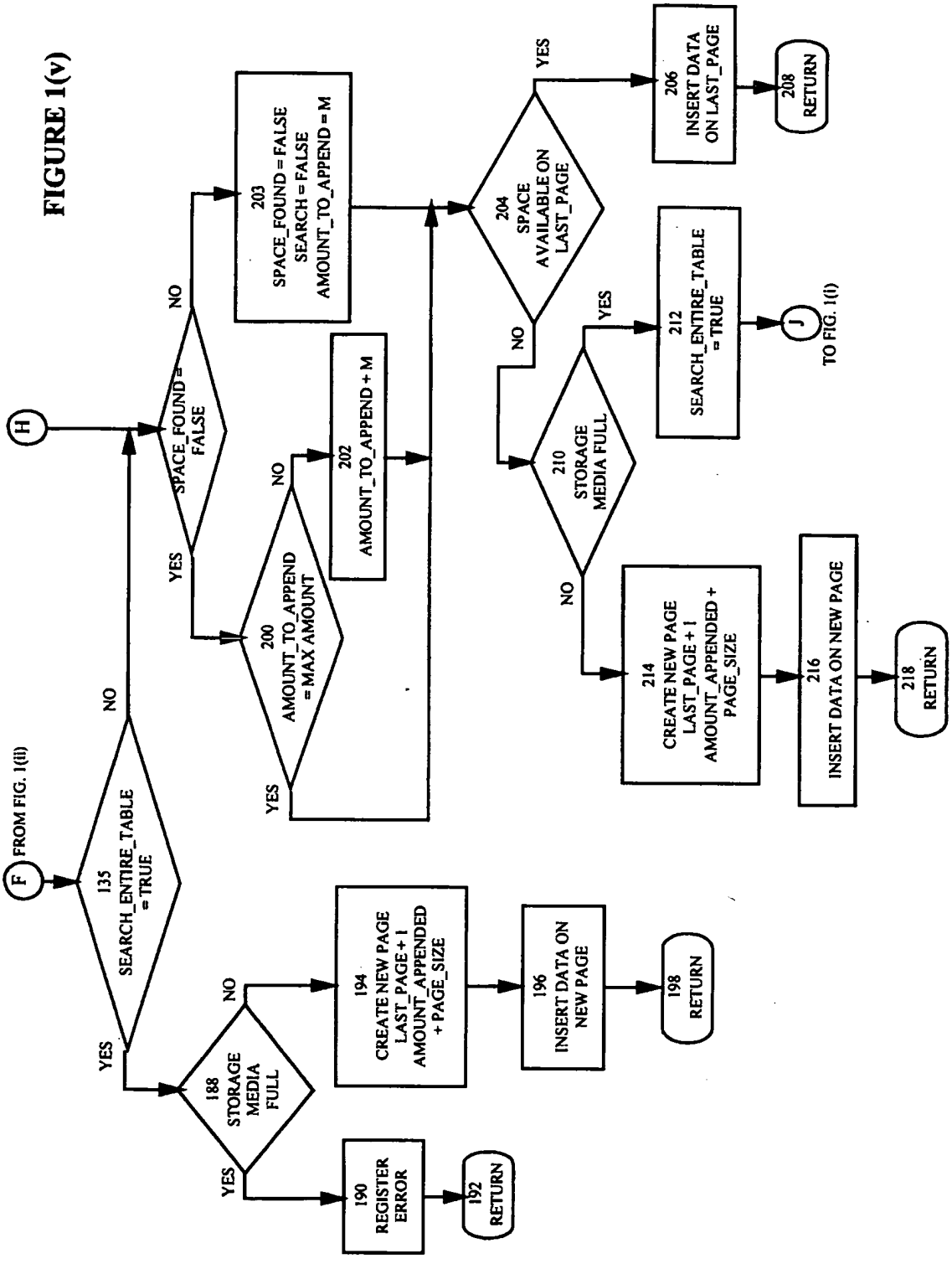
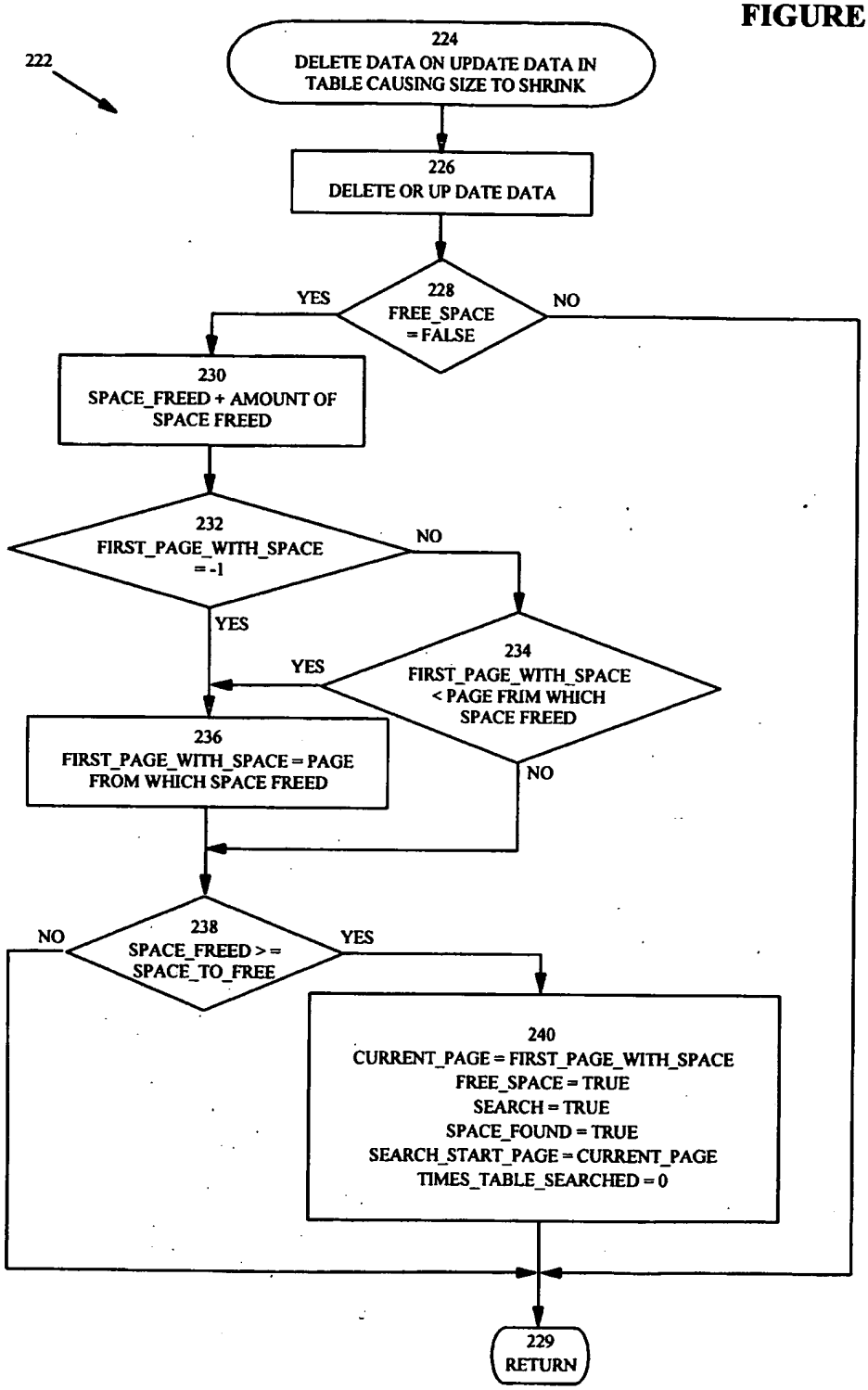

F
FROM FIG. 1(ii)
135
SEARCH_ENTIRE_TABLE = TRUE
YES
NO
188
STORAGE MEDIA FULL
YES
190
REGISTER ERROR
192
RETURN
NO
194
CREATE NEW PAGE
LAST_PAGE + 1
AMOUNT_APPENDED + PAGE_SIZE
196
INSERT DATA ON NEW PAGE
198
RETURN
H
SPACE_FOUND = FALSE
YES
NO
200
AMOUNT_TO_APPEND = MAX_AMOUNT
YES
NO
202
AMOUNT_TO_APPEND + M
203
SPACE_FOUND = FALSE
SEARCH = FALSE
AMOUNT_TO_APPEND = M
204
SPACE AVAILABLE ON LAST_PAGE
YES
206
INSERT DATA ON LAST_PAGE
208
RETURN
NO
210
STORAGE MEDIA FULL
YES
212
SEARCH_ENTIRE_TABLE = TRUE
J
TO FIG. 1(i)
NO
214
CREATE NEW PAGE
LAST_PAGE + 1
AMOUNT_APPENDED + PAGE_SIZE
216
INSERT DATA ON NEW PAGE
218
RETURN

FIGURE 2


222
224
DELETE DATA ON UPDATE DATA IN TABLE CAUSING SIZE TO SHRINK
226
DELETE OR UP DATE DATA
228
FREE_SPACE = FALSE
YES
NO
230
SPACE_FREED + AMOUNT OF SPACE FREED
232
FIRST_PAGE_WITH_SPACE = -1
NO
YES
234
FIRST_PAGE_WITH_SPACE < PAGE FRIM WHICH SPACE FREED
YES
NO
236
FIRST_PAGE_WITH_SPACE = PAGE FROM WHICH SPACE FREED
238
SPACE_FREED > = SPACE_TO_FREE
NO
YES
240
CURRENT_PAGE = FIRST_PAGE_WITH_SPACE
FREE_SPACE = TRUE
SEARCH = TRUE
SPACE_FOUND = TRUE
SEARCH_START_PAGE = CURRENT_PAGE
TIMES_TABLE_SEARCHED = 0
229
RETURN